LAP6 USE OF THE STUCKI-ORNSTEIN TEXT EDITING ALGORITHM

M. A. Wilkes

Washington University

Prepared for:

Advanced Research Projects Agency
Department of Defense
Public Health Services

February 1970

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Computer Systems Laboratory Washington University St. Louis, Missouri	Unclassified
	2b. GROUP

3. REPORT TITLE

LAP6 Use of the Stucki-Ornstein Text Editing Algorithm

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

M. A. Wilkes

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
February, 1970	27 19	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
(1) DOD(ARPA) Contract SD-302 b. PROJECT NO. (2) NIH(DRFR) Grant No. RR-00396 (1) ARPA Project Code No. 655	Technical Report No. 18
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	ARPA – Information Processing Techniques, Washington, D.C. NIH., Div. of Research

13. ABSTRACT

An algorithm which runs on a 2048-word LINC provides efficient on-line editing of character strings virtually unlimited in length. Fixed address LINC tape holds the character sequence in the manner of a scroll. Edited characters are spliced directly in or out of the scroll as it moves across a display scope under the viewer's control. A 512 character "playground" created at the splice point provides sufficient ease to permit changing the scroll contents dynamically, and thereby simplifies several problems commonly associated with on-line editing. Compensatory inserting and deleting are practical. Inserted characters require no special identification and scroll maintenance is automatic. Editing commands and editorial text identifiers are eliminated, and the number of characters which can be inserted anywhere is limited only by the length of the scroll. Line numbers, if provided, are resequenced automatically as the scroll contents change. As little as 2% of the scroll is manipulated in the memory at a time. Despite the relatively slow transfer characteristics of the tape, performance is satisfactory on a LINC for scrolls up to 23 040 characters and is not strongly dependent on the size of the playground.

DD FORM 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Scroll editing						
scope editing						
on-line editing						
tape editing						
LAP6						
LINC						

ia

LAP6 USE OF THE STUCKI-ORNSTEIN
TEXT EDITING ALGORITHM

M. A. Wilkes

TECHNICAL REPORT NO. 18

February, 1970

Computer Systems Laboratory

Washington University

St. Louis, Missouri

ib

Probably the most frequent single cause of error. . .
is the omission of a line (or more lines than one). . . .
  We have been speaking. . of the papyrus roll,
which was practically the only form of book in use in
the Greek world until well into the Christian era.

Frederic G. Kenyon
Books and Readers in Ancient
Greece and Rome

ABSTRACT

An algorithm which runs on a 2048-word LINC provides efficient on-line editing of character strings virtually unlimited in length. Fixed address LINC tape holds the character sequence in the manner of a scroll. Edited characters are spliced directly in or out of the scroll as it moves across a display scope under the viewer's control. A 512 character "playground" created at the splice point provides sufficient ease to permit changing the scroll contents dynamically, and thereby simplifies several problems commonly associated with on-line editing. Compensatory inserting and deleting are practical. Inserted characters require no special identification and scroll maintenance is automatic. Editing commands and editorial text identifiers are eliminated, and the number of characters which can be inserted anywhere is limited only by the length of the scroll. Line numbers, if provided, are resequenced automatically as the scroll contents change. As little as 2% of the scroll is manipulated in the memory at a time. Despite the relatively slow transfer characteristics of the tape, performance is satisfactory on a LINC for scrolls up to 23 040 characters and is not strongly dependent on the size of the playground.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Schwartz has remarked that "the process of editing code online is considered by some to be the heart of a good online system."[1] Unfortunately one can easily observe that it is also considered by some to be remarkably complicated. Editing consists first of finding, preferably visually, an editorial point of interest in a collection of characters. It then entails only two things: adding characters and/or removing characters at that unique point. The inherent simplicity of this activity has been enthusiastically obscured in most computer environments.

A machine such as the LINC[2] is excellently suited to text manipulation. Display scope and keyboard are standard items, as is a two unit magnetic tape transport whose pocket-sized premarked tapes are randomly addressable by block number in a manner similar to the TX-2[3] or the Atlas computer.[4] The LINC searches for a requested block by moving the tape in either direction.

With this equipment it has been possible to realize a "scroll editing" scheme which is notable for its simplicity, directness, and efficiency. Text, or *manuscript*, is held on a LINC tape which is treated as a scroll wound onto the two tape hubs. Information on the part of the scroll over the tapehead is presented on the scope. Motion of the scroll, to expose different parts of the manuscript, is directed from the keyboard.

The concept of scroll editing means that a scroll, theoretically unlimited in length, can be edited anyplace and to any extent by writing directly on the scroll or by erasing information directly from the scroll at the unique points of interest. The scroll remains homogeneous, although its contents, and probably its length, change dynamically with every editorial correction.

Scroll editing is an integral part of the on-line LINC system LAP6,[5] which provides a convenient context for discussing first the operational characteristics and then the editing algorithm.

I. FUNCTIONAL DESCRIPTION

A LAP6 manuscript is any collection of LINC keyboard characters, originally entered by the user at the keyboard or perhaps generated by a program, and retained by LAP6 on a LINC tape as a permanent record of keyboard input.

The Scroll

Manuscripts are retained indefinitely in LAP6 files, but we are concerned here with the "current manuscript" which is held in a reserved *scroll area* on the tape, displayed on the scope, and accessed by the user via the keyboard.

Input characters are saved in a "working section" of the memory. As the section is repeatedly filled, its contents are written in consecutive blocks in the scroll area. A block is simply the transfer unit between tape and memory; the characters carry smoothly across block boundaries, producing a continuous string on the tape. 512 character codes fill one block.

The Display

The scroll is displayed as in Figure 1, which shows five lines of the current manuscript together with line numbers 3412 through 3417. The line numbers are not part of the manuscript. They are supplied automatically by LAP6 and are provided only at the display to help orient the viewer. The last line number displayed, 3417, identifies the "current line."

Figure 1. The Display

Manuscript lines are added to the scroll at the current line and terminated by striking an undisplayed end-of-line key, EOL. The EOL automatically causes the current line number to be incremented.

The display remains centered as the viewer edits, with lines automatically scrolled off or onto the top of the display to compensate for the changes. The frame size is under knob control and is determined only by what the viewer decides is easy to read.

The viewer can do only three things with the scroll: he can move it to look at different parts, he can add lines to it, and he can erase lines from it.

Scroll Motion

The scroll can be moved in either direction at any time to "roll up" the present view and expose a different part of the manuscript, which automatically establishes a new current line. The viewer directs the scroll motion from the keyboard by typing either an arrow and a line number, as in Figure 2(a), or by typing one of four undisplayed key combinations (chosen simply because they lie under the left hand) which move the scroll forward (to a higher line number) or backward, one frame or one line.

(a)   (b)

Figure 2. Moving the scroll from line 3416 to line 340.

Figure 3. Writing on the scroll.   Figure 4. Lines renumbered.

The mechanism itself is less important than the ability to move the scroll easily and directly to other positions. Forward and backward pushbuttons would be more nearly ideal. The line numbers specifically are not required for this operation and are generally used for making only gross adjustments to the scroll position, analogous to finding one's place in printed manuscript by first locating the approximate area by page or chapter number. The exact point of interest is then verified visually based on content. The visual verification is not only preferred by most users, but implies that no further identification, such as explicit reference to line numbers or matched character strings, is required in order to edit the manuscript.

Scroll Editing

Regardless of how the scroll is moved, or whether it is positioned at the end of the manuscript or somewhere in the middle, the display always presents a current line on which changes can be made to the manuscript. If the viewer wants to "insert" lines, he simply types them in. Starting with the situation shown in Figure 2(b), for example, the new lines become line 341, then line 342, etc. (Figure 3). Alternatively, starting again with Figure 2(b), he strikes a delete key to remove line 340. Subsequent deletions will remove line 337, then line 336, etc. The changes are integrated into the surrounding context as he types, and he does nothing further about them.

Within the limits of the total scroll area on the tape, the user can add or delete as many characters as he likes, in any order, wherever he positions the scroll. The manuscript remains one continuous string of keyboard characters accumulated on the tape, as it scrolls in and out of the memory, in exactly the same order as they appear on the scope.

Automatic Renumbering

Lines are automatically renumbered by LAP6 as changes are made to the scroll. The numbers thus remain sequential integers, changing dynamically to reflect the effect of editorial corrections. If following Figure 3 the user backs up the scroll two lines to establish 341 as the current line and adds new information, he will see by locating forward into Figure 4 that the former lines 341 and 342 are now 342 and 343.

II. THE ALGORITHM

Most of the functional advantages of scroll editing derive directly from the editing algorithm which was first proposed by Mishell J. Stucki and Severo M. Ornstein in discussions with the author during March 1965. Despite subsequent refinement the playground concept and the organization of LINC tape and memory to accomodate it are largely their creation.

Although the algorithm is here applied to a character string on a LINC tape, it is appropriate for any randomly addressable storage device such as a disk or, if necessary, core memory. Very little core memory is required, however, and the size of the LINC memory, 2048 12-bit words, greatly motivated articulation of the algorithm. In the LAP6 case the scroll uses a maximum of 768 words of memory. The total manuscript size and manuscript line length are, however, virtually unlimited.

The Playground

The Stucki-Ornstein algorithm is based on the concept of a *playground*, created by separating the character string in the memory at the current line to provide a free space for additional characters coming from the keyboard. In Figure 2(b), for example, the playground has been created between the end of line 340 and the beginning of the undisplayed line 341. Characters, picked up or discarded in the playground, are "spliced" in or out of the scroll as it moves through the memory from one tape reel to the other.

Creating the Playground

Figure 5 shows a manuscript on the tape in blocks 1 through 7 of the scroll area. BS, WS, and CS are 256-word sections of memory corresponding in size to LINC tape blocks. If block 4 contains a line, say line 340, which is requested by the viewer, block 4 is read into the memory and the string broken between lines 340 and 341. In Figure 5, X represents the end of line 340, the "working point," and Y the beginning of line 341, the "continuation point." Block 3 is also read into the buffer section, BS, to provide continuity for the display. The display shows the viewer that the scroll is positioned at line 340, and that the current line number is 341.

The original string is now treated as two separate strings. The first, or *working string*, is contained in tape blocks 1, 2, 3, and the previously mentioned working section, WS, up to point X. The second, or *continuation string*, begins at point Y in the continuation section, CS, followed by tape blocks 5, 6, and 7. The algorithm requires that tape and memory be thought of as continuous between block 3 and the left end of WS, and between the right end of CS and block 5.

The shaded area in Figure 5 represents the playground, an expansion into the memory of the scroll area on the tape. The playground happens to be the same size as a memory section or tape block, but this is not required. It could be larger or smaller, its purpose being only to create "some" extra space at the point of interest.

This organization of tape and memory to separate the string, create a free area between two strings, and define the point X in the memory over which the user has control is basic to scroll editing.
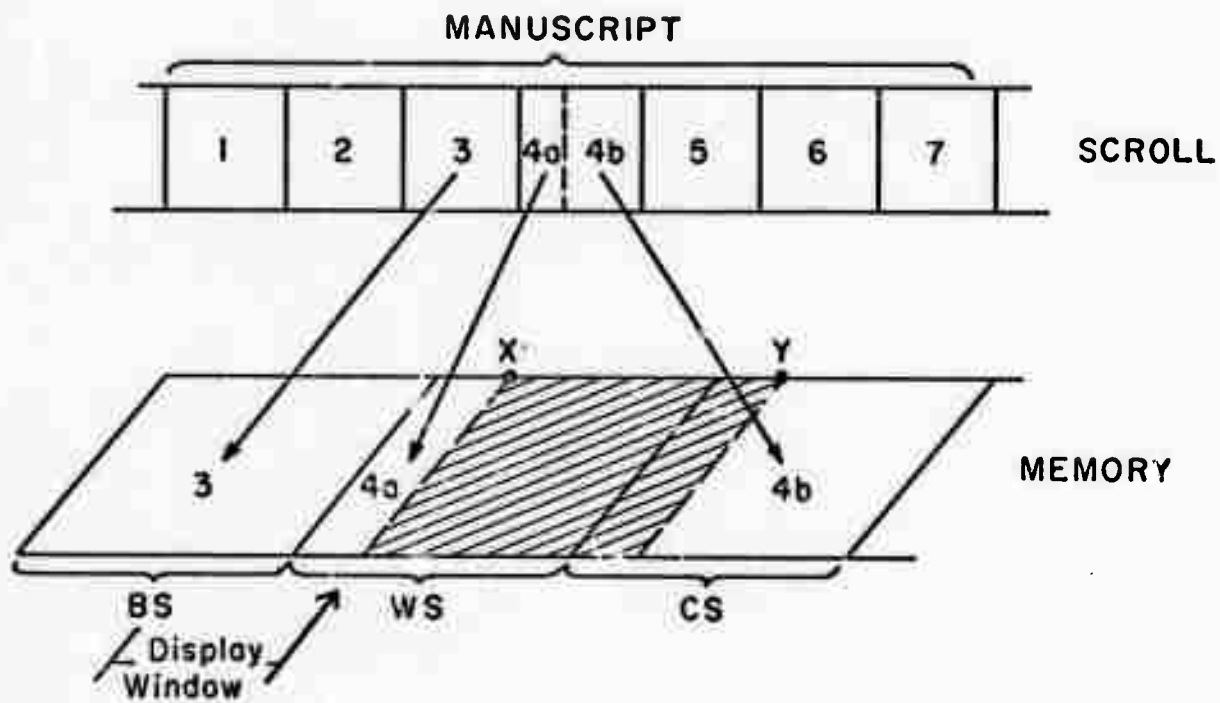
MANUSCRIPT

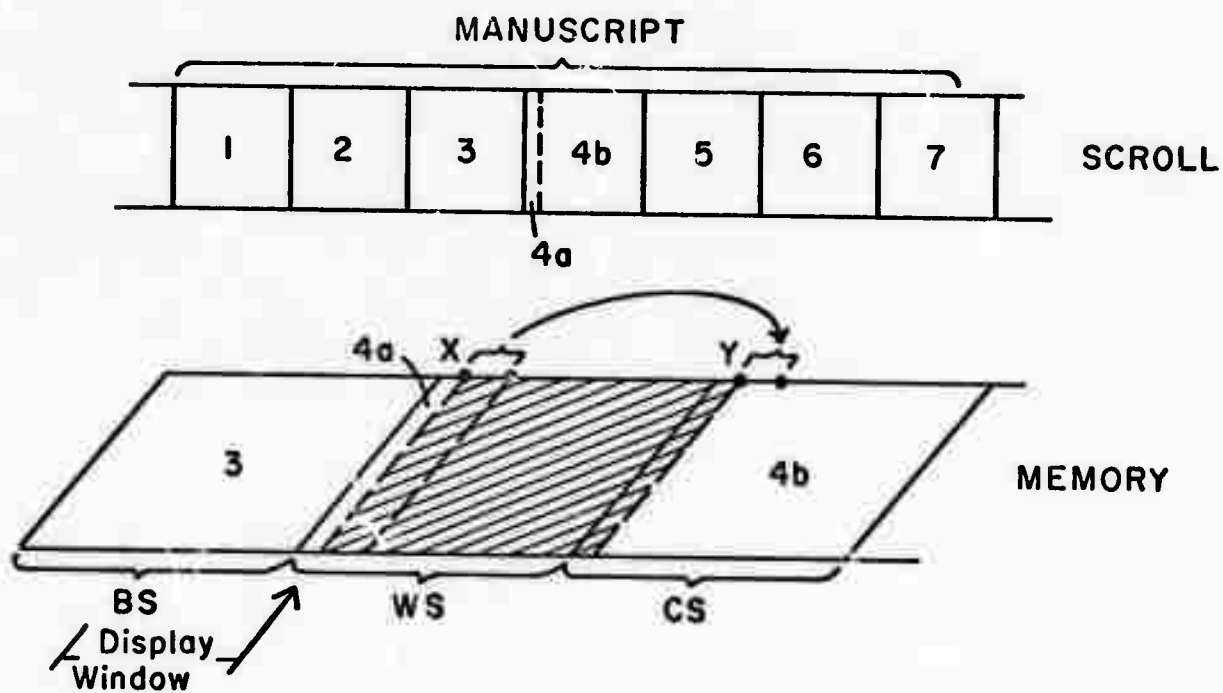Figure 5. Creating the playground

MANUSCRIPT

Figure 6. Locating backward

Positioning the Scroll

If in Figure 5 the viewer repositions the scroll to, say, line 330, so that the scroll must be moved backward, characters are transferred from the working section into the continuation section (Figure 6), until the point X is at the end of line 330. Y is now at the beginning of line 331, and lines 331-340 are now at the beginning of the continuation string. The playground has simply shifted left. Locating forward is identical, except that the playground is shifted to the right.

Adding and Deleting Characters

The primary purpose of creating the playground is, of course, to be able to handle new information. Additional characters are stored in the playground starting at the working point as they are typed. X, and the display, move into the playground the appropriate amount (Figure 7) keeping track of the end of the working string. Adding new information shrinks the playground. Other than that, there is nothing special about the new characters. As can be seen in Figure 7, the working string is still continuous from block 1 to point X.

Likewise, if the user deletes lines,* X moves the appropriate amount to the left, shortening the working string. Deleting increases the size of the playground.

One obvious advantage of the scheme is that adding and deleting are compensatory. When initially created, the playground can accomodate 512 characters.** The total numb   characters which can be added before a new playground must be created is therefore 512 *plus* the number deleted.

In either case, since the changes are made exactly where they belong in the string, no further manipulation is required by either the user or LAP6 in order to incorporate them. The relationship of the working string to the continuation string remains well defined, and there is no need to distinguish an "edited" from an "unedited" manuscript.

The essential elements of the algorithm can be summarized:

1. Moving the scroll in either direction shifts the playground left or right, moving X and Y in parallel.

2. Adding or deleting information causes the playground to shrink or expand, since only X is moved. Point Y and the continuation string are temporarily ignored.

In what follows, combinations of these two activities are considered in connection with the line numbers, section boundary conditions, and the full playground situation.

---

*LAP6 is line-oriented; the algorithm is the same for character-oriented editing.

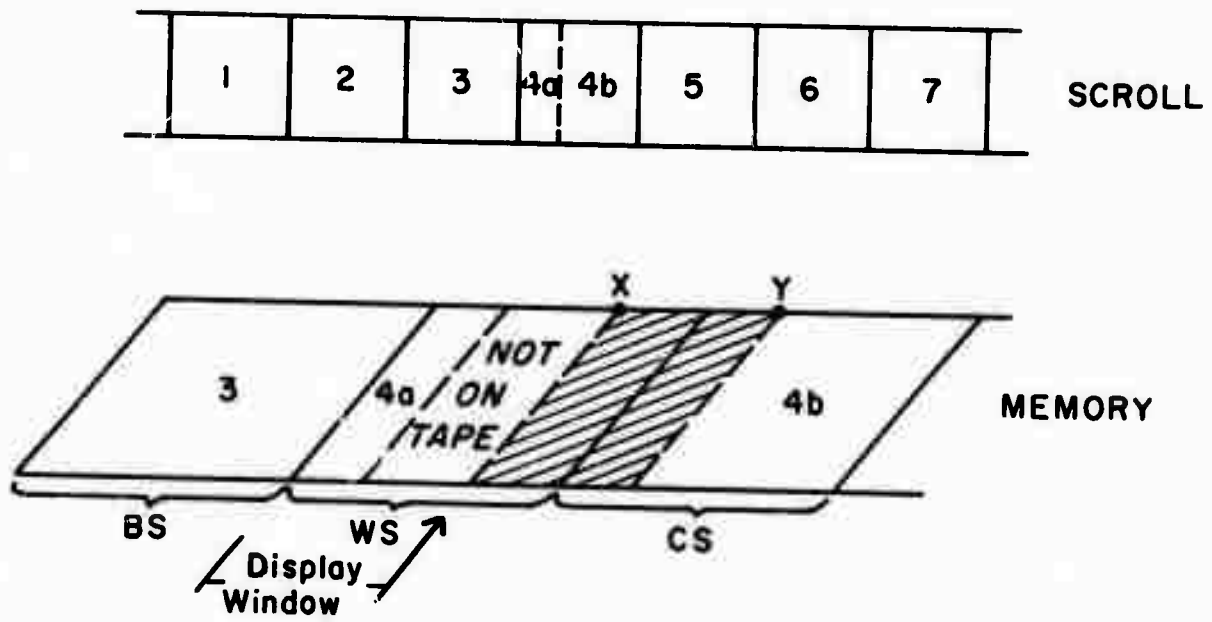**Two character codes are stored per word.
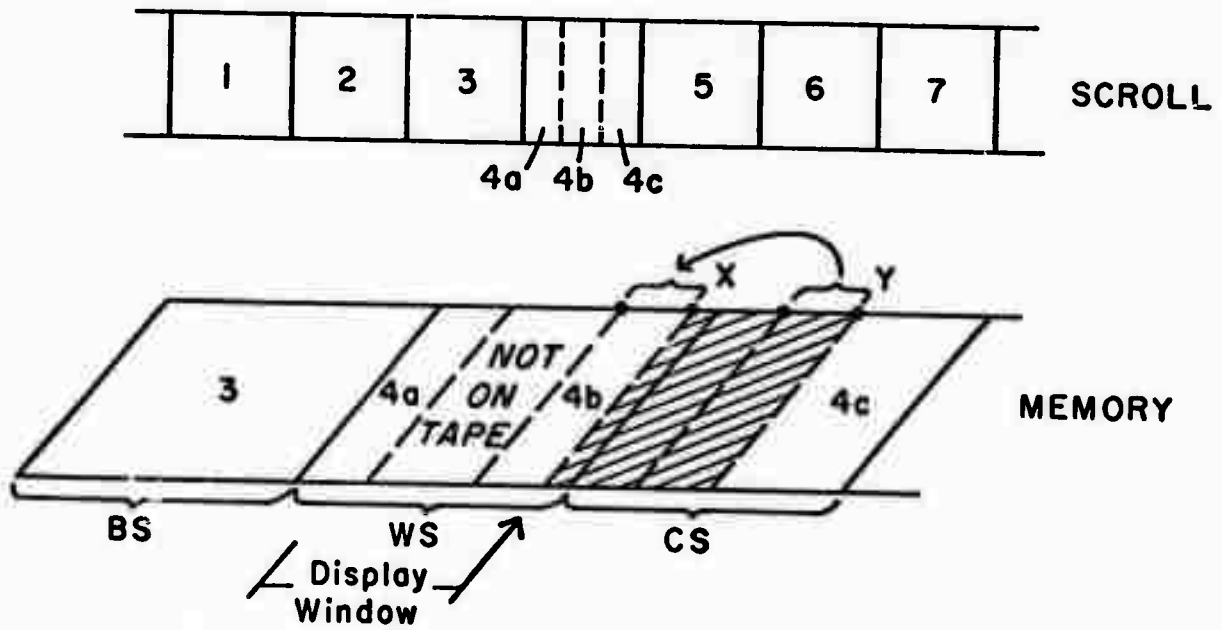
Figure 7. Adding new information

Figure 8. Locating forward and splicing

Locating and Editing

Figure 8 shows the new information added in Figure 7 being spliced into the working string by moving the scroll forward. The part of the string originally contained in block 4 is now reconfigured in the memory in three pieces (4a, 4b, 4c); only the working section contains the correct continuation of the string from block 3.

Since locating simply shifts the playground, the size of the remaining playground is the same in Figures 7 and 8. Thus, although the playground may eventually be filled, the added characters can be distributed throughout the scroll. It follows also that changes made to one part of the scroll may be compensated by changes made to a different part.

Numbering the Lines

The line numbers are represented simply by two counters. One monitors the current line number, the other the number of the last line on the scroll The trivial numbering scheme is possible because scroll editing does not require that the lines have fixed identifiers.

When the scroll position is changed, the current line number counter is incremented or decremented. When information is added or deleted, both counters are incremented or decremented. When the contents of the two counters are equal, there is, by definition, no continuation string.

The current line number counter is incremented, therefore, whenever lines are added to the working string, without regard to the source of the added information. Thus, as keyboard input is interspersed with forward locating, (Figure 8), the "old" lines are automatically renumbered as they are brought from the continuation string onto the scope.

Section Boundaries

When either X or Y reaches a section boundary, the section is either full or empty. In order to continue the current operation the tape must be written or read.

The working section will be filled (X at the WS-CS boundary) either by locating forward or adding new information. When this happens (for example, continuing to locate forward following Figure 8), WS, since it contains the continuation from block 3, is written in block 4 (Figure 9). WS is then transferred into BS, and X is reset at the BS-WS boundary. This resetting keeps the working string continuous between blocks 1, 2, 3, 4, and X in WS. It is not to be confused with increasing the size of the playground, which happens only when deleting.

The shaded area in Figure 9 thus represents only the size of the remaining playground; the playground itself does not leave the working section. (Permitting X to continue into CS requires for some manipulations that CS and WS switch roles, which is programmatically awkward.)

Only locating backward can fill CS. Following Figure 8 again, this puts Y on the WS-CS boundary (Figure 10), and the continuation section, since it continues into block 5, is written in block 4. Y is reset at the right-hand boundary of CS.
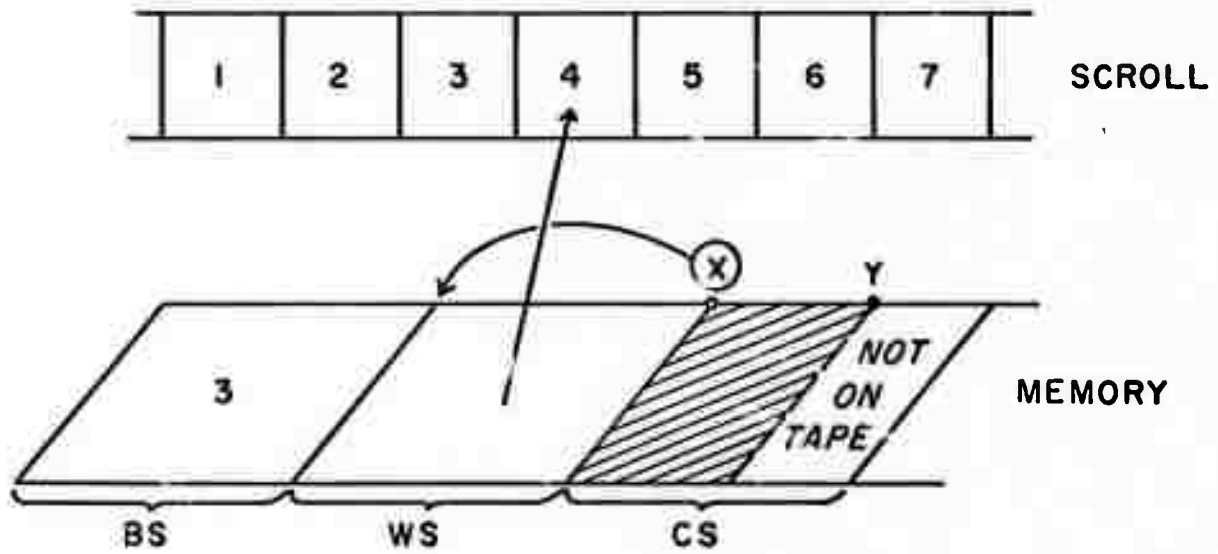
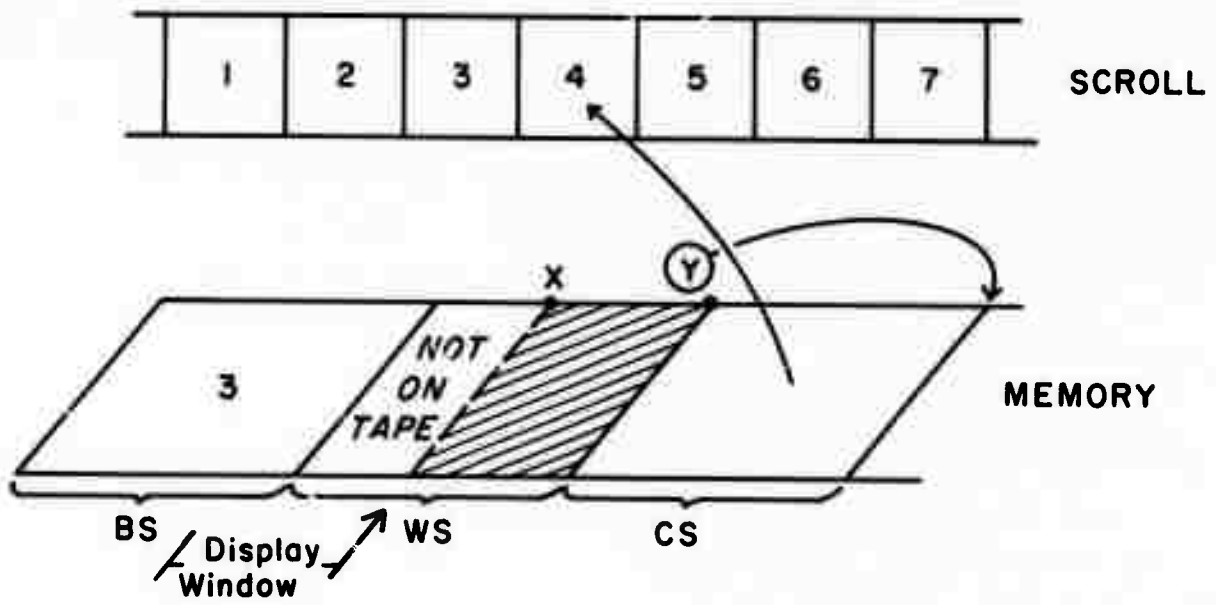Figure 9. Working section full

Figure 10. Continuation section full

In either Figure 9 or Figure 10 the original contents of block 4 are, of course, destroyed. However, because block 4 was initially read into the memory, its space on the tape is free and no information is lost. The memory contains the same *amount* of information "NOT ON TAPE" that was entered as "new" information in Figure 7, and consequently the size of the playground is still the same.

The information not on the tape is therefore not necessarily inserted information. In Figure 9, for example, it is the remainder of the 4c segment of the original contents of block 4. As the user works with the manuscript, the tape blocks thus contain different images of the two string sequences at different times.

When the tape transfer is made, the display is interrupted for about 0.1 second. Since block 4 was the last block read (Figure 5), the tape is properly positioned and there is little travel time loss.

Either deleting or locating backward can leave the working section empty. If, starting with Figure 5, information is deleted until X lies on the BS-WS boundary, that part of the string is no longer in the manuscript and the playground is effectively increased by that amount. Since the deleted characters are simply characters which need not be written back on the tape, the additional playground created is represented by the shaded area on the tape in Figure 11. Since WS is a continuation from block 3, block 3 is read into WS (and block 2 into BS). X is reset at the WS-CS boundary, and the working string is now contained only in blocks 1, 2, and WS up to point X. The size of the playground, now marked by the transfer of block 3, is, of course, 3½ (from Figure 5) plus the number of characters in 4a.

Similarly, had CS been emptied by locating forward, since its string is continued in block 5, block 5 would have been read into CS and Y reset at the WS-CS boundary, making the continuation string begin at point Y in CS followed by blocks 6 and 7.

If the user continues to delete following Figure 11, the two strings can become widely separated, as shown by the shaded area on the tape in Figure 12. The working string is now continuous between block 1 and WS. Figure 3 shows what happens if at this point the scroll is moved backward until the continuation section is full. As CS is written on the tape in block 4, segment 2b is moved from block 2 to block 4, and the additional playground is shifted left into blocks 2 and 3. The shift is identical to the playground shift in Figure 6, but it is done on the tape.

Joining the Strings

If the scroll is moved forward following Figure 12, WS will eventually be written as information moves through the memory from block 4 to block 2. Clearly, locating at the last line is all that is required to join the two strings. When the last character is moved from Y to X, the working string is the entire string and the playground is "pushed off" the end.

The LAP6 user is never required to join the strings. He states system commands at any time, regardless of the position of the scroll or the extent of editing. If the strings need to be joined (e.g., to file the current manuscript), LAP6 joins them automatically. This avoids unnecessary tape handling since under program control the decision to join the strings can be based only on the extent of editing, and not influenced by the scroll position. In any case, no scroll manipulation of a maintenance sort is ever required of the viewer.
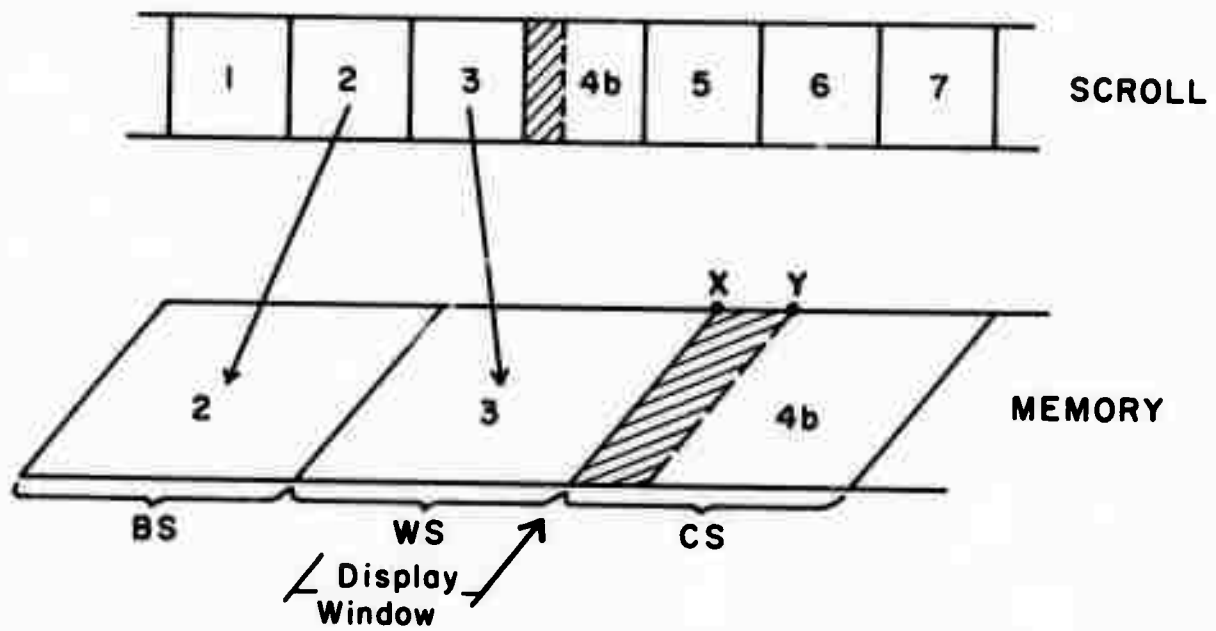
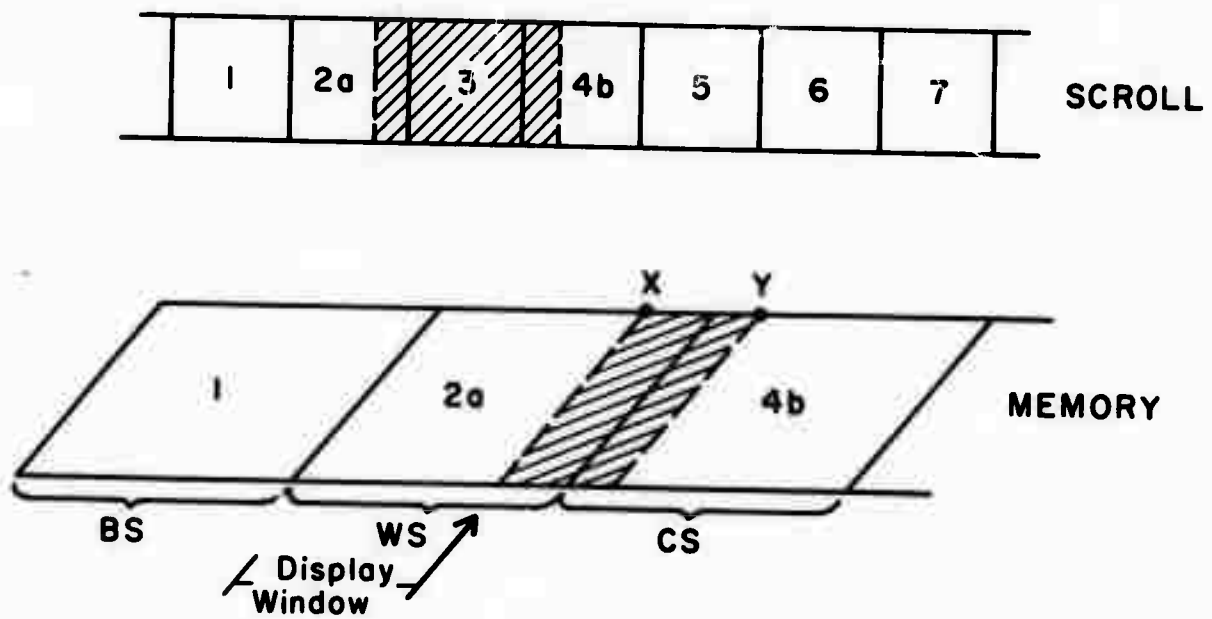Figure 11. Working section empty and playground expanded

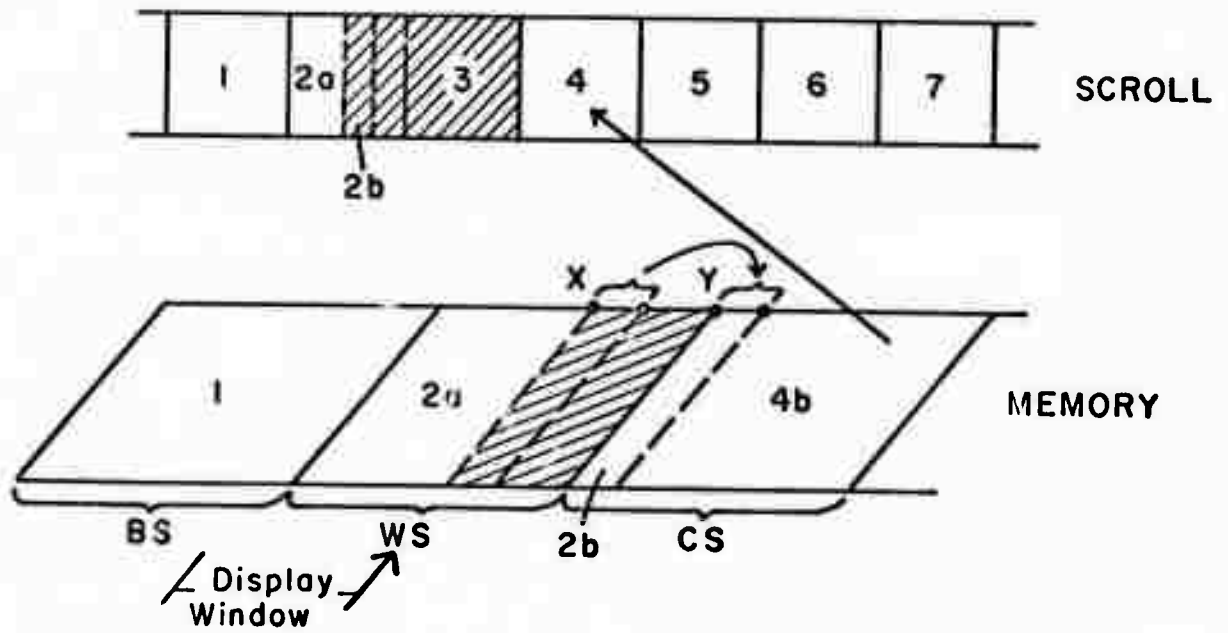Figure 12. Strings separated by deleting
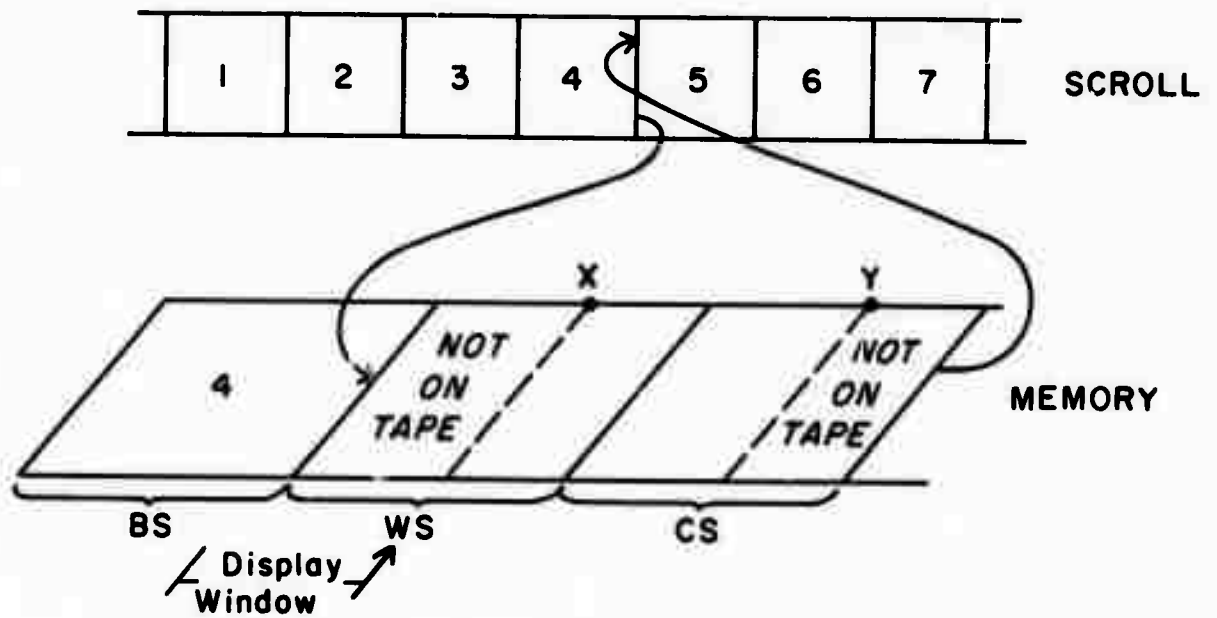
Figure 13. Playground shift

Figure 14. Playground full

Playground Full

The playground is full when 512 characters in the memory are NOT ON TAPE, and when the last *tape* block of the working string is contiguous to the first *tape* block of the continuation string. The situation of Figure 14 follows that of Figure 9 by adding enough characters to total 512. Since WS is the continuation from block 4, and block 5 continues from CS, the blocks are contiguous.

At this point information about the current scroll position is saved and the two strings are joined automatically as described above. A new playground is then created at the former scroll position. The result on the scope and in the memory is identical to Figure 14, although the segments formerly NOT ON TAPE would now be labelled "5a" and "5b" (as for Figure 5). CS now continues into block 6. As far as the viewer is concerned there is no change.

It must be emphasized that a full playground in no way represents an awkward situation or interferes with the system operation. The procedure for handling it is identical to executing two locate requests, except that it is automatic. The programming overhead incurred is therefore minor; in the LAP6 case it is about thirty machine instructions.

III. EFFICIENCY

The standard configuration of LAP6 has a 45 block scroll which accomodates, at 512 characters per block, 23 040 characters. The size has been found more than adequate for most LINC applications, although a few have used scrolls as long as 270 blocks. Although the algorithm is unrelated to scroll size, its efficiency, of course, is not.

Moving the scroll through the memory as called for by the algorithm requires a little less than one second per tape block. This includes reading the block into the continuation section, writing it back onto the scroll from the working section, and checking the transfers. The write and check operations, for which the LINC has to reverse the tape twice, are the most costly. The LINC tapes are read at regular travel speed of 23 blocks per second.

In practice, the user is aware of frequent but brief tape motions. Long delays are infrequent, due to a variety of influences which will be discussed. In any case, our experience in the LINC environment has shown that being able to see the tape move makes any delay more tolerable.

Editing Efficiency

There are no editing commands, which reduces both typing time and errors, and the explicit activity of adding and deleting characters probably cannot be further simplified. The response on the scope is always instantaneous. Since the scroll is properly positioned this is true even when the user edits across a boundary of the working section. A delayed response is possible in the editing situation only if the playground fills up.

Playground Efficiency

Although the user is unaware of the playground *per se*, he may notice some extra tape motion if it fills up, depending on how far the current line is from the last line.* In most situations, however, the playground never fills up, despite extensive editing, because of the compensatory deleting effect. Some users have never observed it.

For the same reason the size of the playground when initially created is not critical. Since delays due to a full playground are seldom incurred, increasing the size of the playground does not significantly affect the general efficiency.

Reducing the playground's size, however, will eventually degrade performance. Although it is difficult to determine a smallest tolerable size, the playground should clearly not be smaller than a typical manuscript entry, i.e., than the least number of characters which might be inserted before any are deleted. Preferably it should be a multiple of that number. In LINC program preparation, for example, a manuscript entry is an instruction line of typically ten characters. Our experience with the 512 character playground indicates that

---

*The LAP6 scroll is "open-ended" only in the direction of the last line. Efficiency can be somewhat improved by "smoothing" the scroll in either direction, whichever distance is shorter. On the one-tape LAP6 system overall tape allocation is more efficient if one end of the scroll is fixed.

this multiple of about 50 times the size of an entry is probably more generous than necessary. When the data are mailing lists or bibliographies for which a typical entry is about 100 characters, performance, although poorer, is still generally acceptable to most users. Acceptability, however, is undoubtedly influenced by the fact that a full playground still requires no user action, except waiting, in order to continue editing. For most situations the author would not recommend using a playground smaller than five times the size of a typical entry.

The worst case situation regarding the playground occurs when a number of characters is added, and none deleted, near the beginning of a long manuscript. Using a LAP6 "Add Manuscript" command, for example, a filed manuscript can be added to the scroll manuscript at the current line. This amounts to inserting one manuscript in the middle of another, but the playground fills up once per block of incoming manuscript. Provision is made for the LAP6 user to break the scroll into separate manuscripts, i.e., temporarily eliminate the continuation string, first.

Locating Efficiency

To reduce the delay involved in repositioning the scroll, LAP6 uses one of two techniques to move the scroll. The first moves it through the memory at about one second per block as described. The other uses a "scroll index" to determine the block number of the scroll block containing the requested line. It then moves the tape, at 23 blocks per second, directly to the required block.

The scroll index is a table of line numbers, each entry occupying one memory word corresponding to a scroll block. Every time the contents of the working section are written on the scroll in block n, the current line number is recorded in the nth word in the index. The location of a particular line can later be determined by comparing pairs of index entries.

This second technique is used whenever moving the scroll through the memory will not change the string sequence on the tape. This is the case when there is no continuation string, or when no editorial changes have been made since the working and continuation strings were last joined. The 23 to 1 saving is considerable when the viewer is simply locating and reading, as, for example, when first finding his place in the manuscript.

In the algorithm as presented here the playground is external to the character string on the tape, that is, it is created without disturbing the scroll sequence. Although the temptation exists to embed the playground physically in the string, this eliminates the possibility of ever using the faster technique to move the scroll. The two strings thus separated will always have to be rejoined, even when no editorial changes have been made. In the same manner the automatic rejoining of the strings described earlier will have to be based on scroll position alone.

If the playground is externally created, however, one scroll position can simply be abandoned in favor of another. A new playground is then created at the destination.

The combination of techniques keeps the locating delays from dominating the editing situation. If the user is only reading, the faster technique is used automatically. If he is editing, he tends to spend more time typing than he does moving the tape, despite the fact that the tape will in this case move at the slower rate.

Changing one character in the first block of a thirty block manuscript, then, will cost the user about thirty seconds when LAP6 rejoins the two strings. On the other hand, if extensive changes are made in the same manuscript, a process which may take the user several hours, the sum of all the locating delays will, if the locating pattern is sequential, still be only thirty seconds.

CONCLUSIONS

The efficiency of scroll editing is primarily influenced by the transfer characteristics of the scroll device, rather than by limitations inherent in the algorithm. Even with a relatively slow device, however, the balance between the amount of data which can be efficiently handled and the amount of core memory required is excellent. Using a LINC tape, a ratio of total scroll size to transfer unit size of 45 to 1 is reasonably efficient. Further, the 45-block scroll size is not itself a limitation in most applications of the 2048-word LINC.

In situations where a longer scroll is needed, one would recommend either a device with faster transfer characteristics, such as a disk, or an increase in the size of the transfer unit. The total scroll area can be quadrupled, for example, without significantly reducing the efficiency, by making the working and continuation sections in the memory each the equivalent of four tape blocks instead of one. This is possible on some of the larger memory LINCs now available.*

Surprisingly, perhaps, the concept of creating "some" working space, – a playground, – in the middle of a character string presents no special problems. The algorithm can be programmed in about 700 LINC instructions,[6] including the display, and the investment required specifically to handle the playground itself is trivial. Nor is the efficiency of the algorithm especially sensitive to the playground's size. In practice the algorithm has, if anything, performed better than anticipated.

Scroll editing has been in routine use by about 2,000 people on all varieties of LINCs since the summer of 1967. The technique has been found excellent for extensive editing of long character strings, and the resulting manuscript structure is simple and straightforward. As a result it is used for data preparation in a variety of applications including the preparation of chemical formulae for molecular graphics work, bibliographies for information retrieval systems, flow charts, and a variety of programming languages.

The popularity of scroll editing is as much attributable to its operational ease as to its efficiency. The elimination of scroll maintenance and editing commands have minimized the user's responsibility, and the editing process itself is direct, reliable, and unencumbered.

*The micro-LINC 300, LINC-8, and PDP-12 all have memories expandable to 32,000 words.

REFERENCES

1. J.I. Schwartz, "Online programming," *Comm. ACM,* vol. 9, pp. 199-202, March 1966.

2. W.A. Clark and C.E. Molnar, "A description of the LINC," in *Computers in Biomedical Research,* vol. 2, R.W. Stacy and B. Waxman, eds., New York: Academic Press, 1965, pp. 35-66.

3. R.L. Best and T.C. Stockebrand, "A computer-integrated rapid-access magnetic tape system with fixed address," *Proc. Western Joint Computer Conf. 1958,* New York: American Institute of Electrical Engineers, 1959, pp. 42-46.

4. T. Kilburn, R.B. Payne, and D.J. Howarth, "The Atlas supervisor," *Computers: A Key to Total Systems Control, 1961 Eastern Joint Computer Conf., AFIPS Proc.,* vol. 20, New York: Macmillan, 1961, pp. 279-294.

5. M.A. Wilkes, *LAP6 Handbook,* Computer Research Laboratory Tech. Rep. No. 2, Washington University, St. Louis, May 1967.

6. M.A. Wilkes, *LAP6 Manuscript Listings,* Computer Systems Laboratory, Washington University, St. Louis, May 1967.